POLITECNICO

MILANO 1863

Computer Science and Engineering

PowerEnjoy Service - Design Document

December 7, 2016

Prof. Luca Mottola

Authors:

- ZHOU YINAN(Mat. 872686)
- ZHAO KAIXIN(Mat. 875464)
- ZHAN YUAN(Mat. 806508)

Contents

1 INTRODUCTION

1.1 Purpose

The Design Document serves to describe the structure of the PowerEnJoy service. It provides all the detailed information for building the system. More precisely, it provides the information on the chosen architecture style and design pattern. It explains how the components are implemented and interacted in the system. The document also provides the verification of fulfilling the requirements listed in the RASD documents. In general, the document is served as a guideline for program developers.

1.2 Scope

The Design Document shows how the system is built and explains how the functional requirements in the RASD file are realized. The document covers high level architecture design, interacting components, algorithms and user interface design.

1.3 Definitions, Acronyms, Abbreviations

- RASD : Requirement Analysis and Specification Document

- DD : Design Document

- MVC : Model View Controller

- REST : Representational state transfer (REST) or RESTful web services are one way of providing interoperability between computer systems on the Internet.

- JSON : JSON (JavaScript Object Notation) is a minimal, readable format for structuring data.

1.4 Reference Documents

- Specification Document Assignments AA 2016-2017

- RASD

1.5   Document Structure

The Design Document is divided into 7 parts :

- **Introduction** : This section introduces the structure of Design Document and some basic background knowledge to understand the document.

- **Architecture Design**
  1. High level components and their interactions : This section gives a general description of how the components are defined and how they communicate with each other
  2. Component view : This section gives detailed information of components defined in the system
  3. Deployment view : This section describes how the components are deployed in order to act correctly
  4. Runtime view : This section gives the sequential diagrams of how the users accomplish their requests
  5. Component interfaces : The interfaces of components are described in this section
  6. Selected architectural styles and patterns : This section explains the reason of choosing certain architecture and the benefits accompanied
  7. Other design decisions

- **Algorithm Design** : In the section, the necessary code samples are given in order to clearly demonstrate the component interaction

- **User Interface Design** : this section presents mockups and user experience explained via UX and BCE diagrams

- **Requirement Traceability** : This section shows how the requirements in the RASD are accomplished in the design

2 ARCHITECTURAL DESIGN

2.1 High level components and their interaction
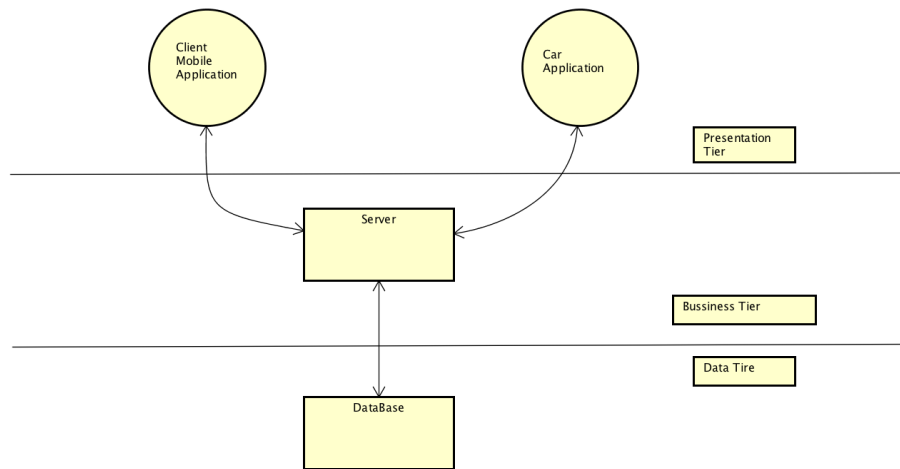
Here we choose to use the 3 tier architecture: The Presentation tier includes the Client Mobile Application, Client Web Browser and the Car Application; The Business tier includes the Server; The Data tier includes the database.

Here a MVC pattern is also used. The presentation tier is responsible for all the GUI part, thus it is the view part. The server maintains all the business logic and the database is responsible for data manipulation. Thus the model and controller are in the server and database side.

2.2 Component view
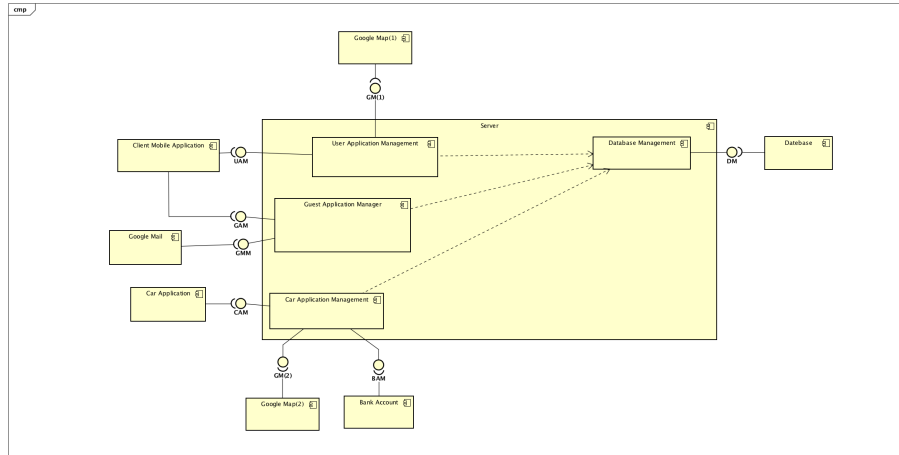
For the Server, there should be several interfaces, which are in charge of communicating with other components:

- **User Application Manager** : Component that provides the interface for the users of the system and the Google Map. User Application Manager should provides the options of checking the available car around the specified position, reserving car and canceling the reservation. With the help of the Google Map, the system can know the location of the user.

- **Guest Application Manager** : Component that provides the interfaces for the unregistered guest of the system and the Google Mail. It provides the option to create an account. Finishing the register, the system would send an e-mail to the guest through the Google Mail.

- **Car Application Manager** : Component that provides the interface for the car Application, Google Map and Bank Account. The Car Application Manager can know the location of the car, charge the specified bank account and send the instruction to the car.

- **Database Manager** : Component that provides the interface for the Database for collecting the information which are collected by other components.
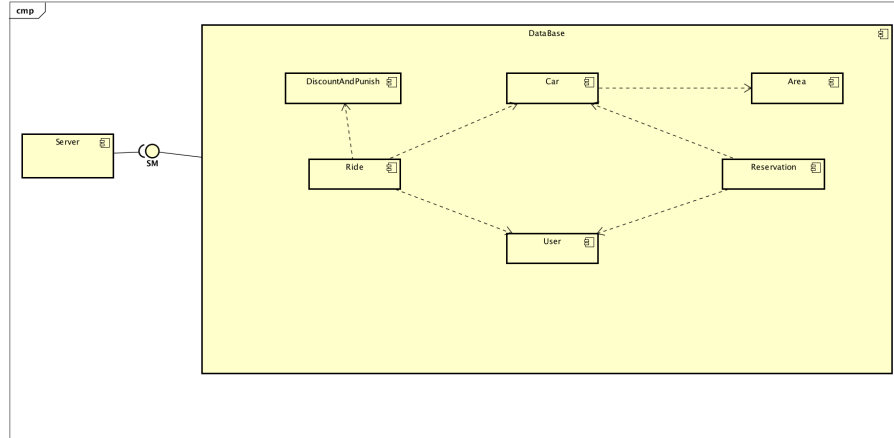
For the Database, there are several component for storing the data of the system. The Database should provides the interface of the Server for information exchange. Through the interface, the Server can transmit and store the information in the Database. The Server can also retrieve the information from the Database according to the corresponding instruction.

- **Car** : Component that stores the information of the cars. The Car should stores the position, plate, battery, capacity, state and the currentPosition of each car.

- **User** : Component that stores the information of the users. The User should stores the credential and the paymentInfo of each user.

- **Ride** : Component that stores the information of the riding. The Ride should stores the riding information, including the renting car, corresponding user and so on.

- **Reservation** : Component that stores the information of the reservation. The Reservation should stores the user and the corresponding car of each reservation.

- **DiscountAndPunish** : Component that stores the information of the discount and punish of each ride. For each ride, when finishing the riding, the DiscountAndPunish component should provide the information about the discount and the punish, for calculating the total charging.

- **Area** : Component that stores the area of each position.

2.3 Deployment view
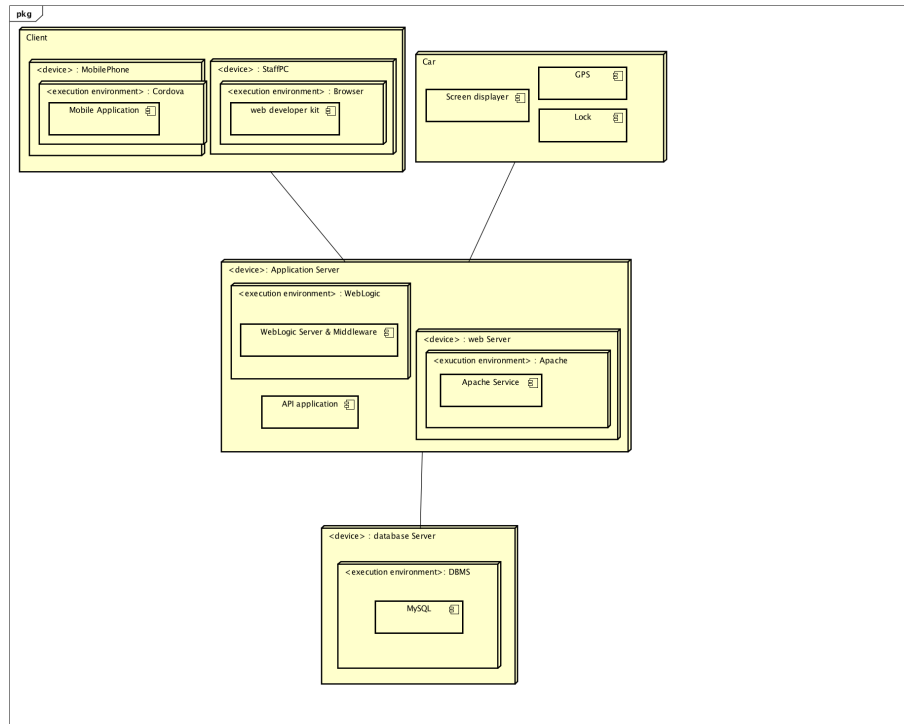
2.4   Runtime view

2.5   Register
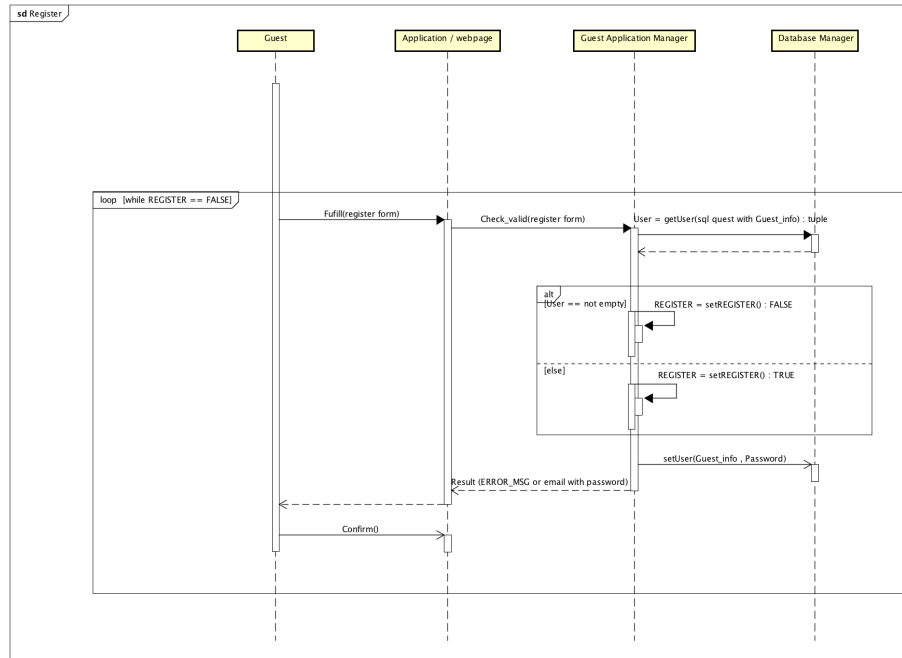
The sequence diagram shows procedure of registration. It starts when a guest request the registration on the client application. After guest fulfilling the register form , the information will be sent to the server, where guest application manager checks for validation. It accesses, through the database manager, the database and checks if the information already exists. In the negative case, the serve sends back to application interface a message with error; in the affirmative case,the server allows the database to memorize the guest's credential and password, and Guest Application Manager will send back an email to guest with Password created.

2.6 Login
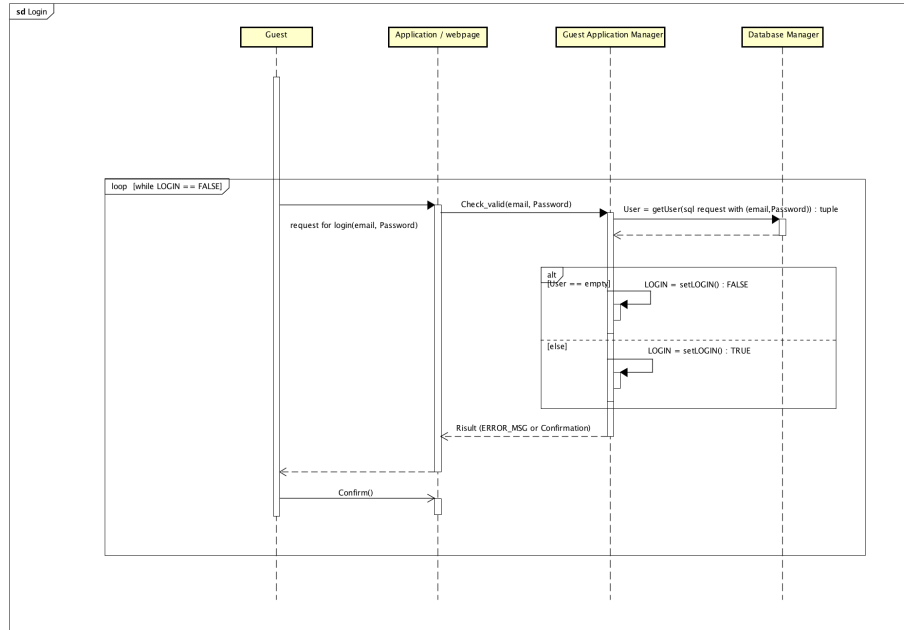
This sequence diagram shows the procedure of users logging into the system. The Client Application sends to the server the user's email and password to check for correctness.If a corresponding tuple exists in the data base, the Guest Application Manager sends back the user information in the format of JSON;If not, the server sends to client application an error message.

2.7 Reserve
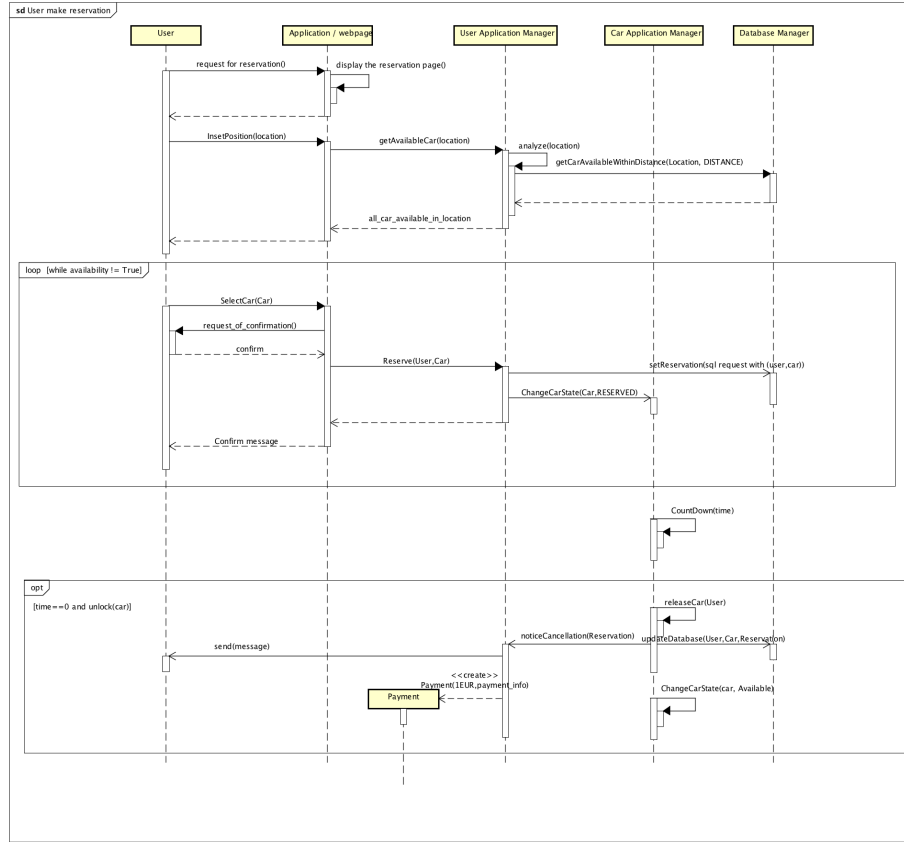
This sequence diagram shows how a user can reserve a car. First the user inserts a position where he wants to start the ride,it can either be inserted by user or generated by GPS automatically. This information will be sent to User Application Manager, it gets all the car information form data base and selects certain cars within nearby distance. Then the list of cars will be sent back. And the Client application, after user confirming the selection,notifies the User Manager. The User Manager then calls Database Manager to insert into the database the reservation information. Finally a message will be sent to application, notifying the user the successful reservation.

The reservation lasts for one hour. At the end of this time, the reservation will be canceled. the server updates the reservation state, changes the car state into Free,charges 1EUR , and at last sends an email to notify

the user the cancellation and punishment fee.

2.8   Pick up
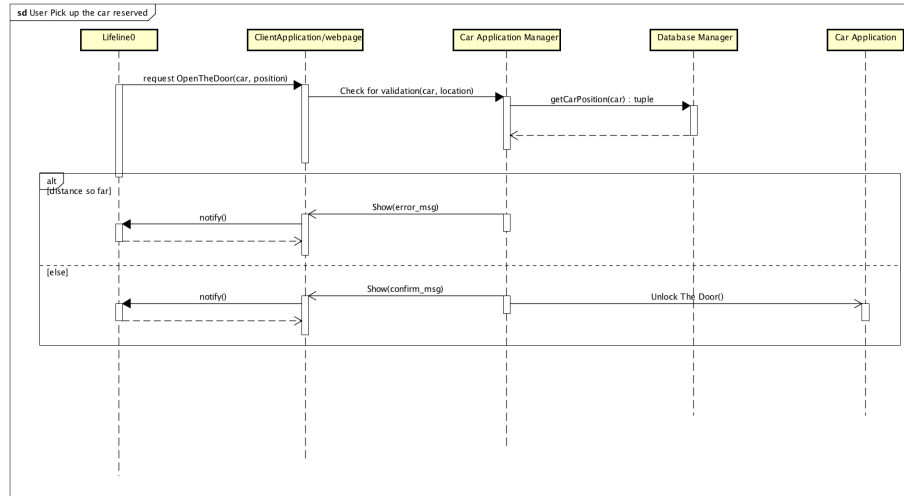
When the user is near the car, he send his current location to notice the server. Car Manager then verifies whether the position received is near enough to unlock the car. It sends back a message with positive or negative confirmation based on the result of verification. Moreover Car Manager sends a command to the corresponding car in order to unlock it.

2.9 Renting
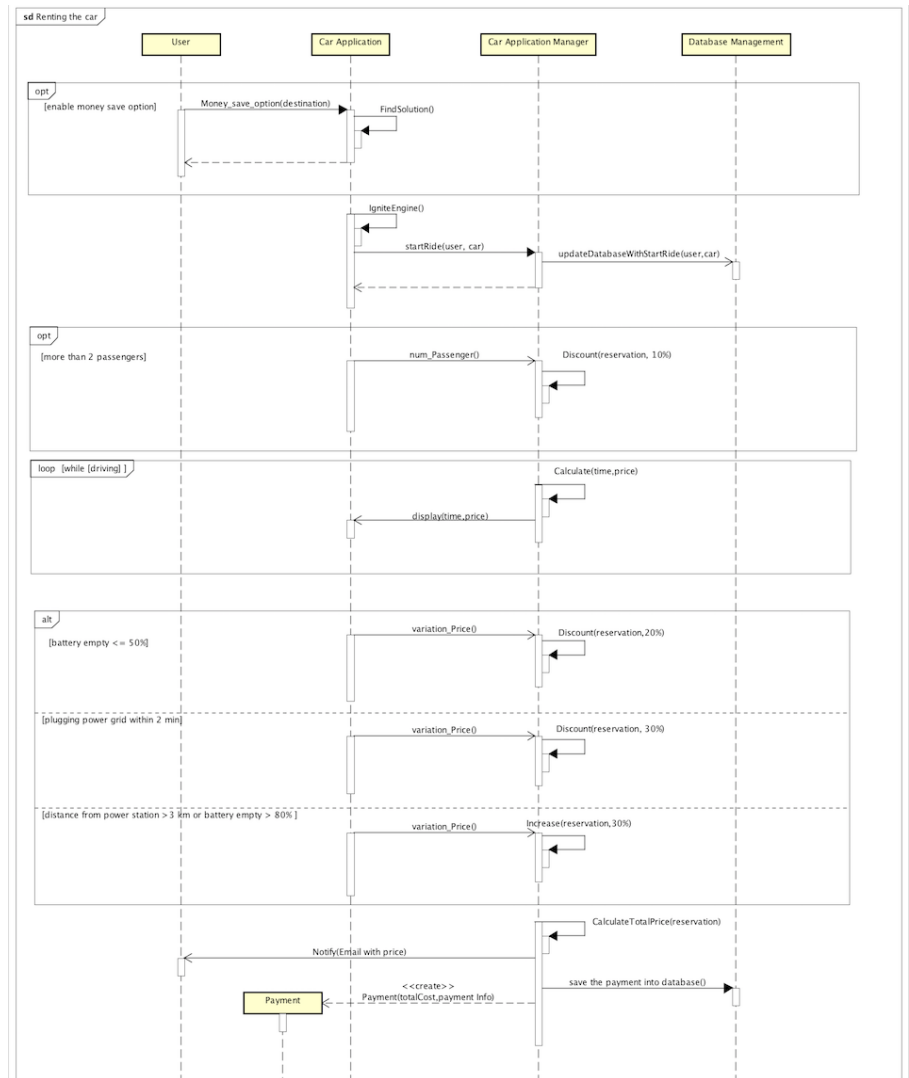
The sequence shows how to start a ride. User can optionally choose to enable money save option or not. After Ignition of the car engine, there are some sensors in the car that can detect how many passengers are there. If it's more than 2 people (exclude user), the Car Manager will apply 10 percent of discount on this ride. During the ride a display shows the information(time ,cost) of the journey. There are some other conditions for discount based on the state of car after the ride. The serve checks which one is presented and applies the corresponding discount. The total price will be calculated by Car Manager and sends back to Client application.

2.10 Component interfaces
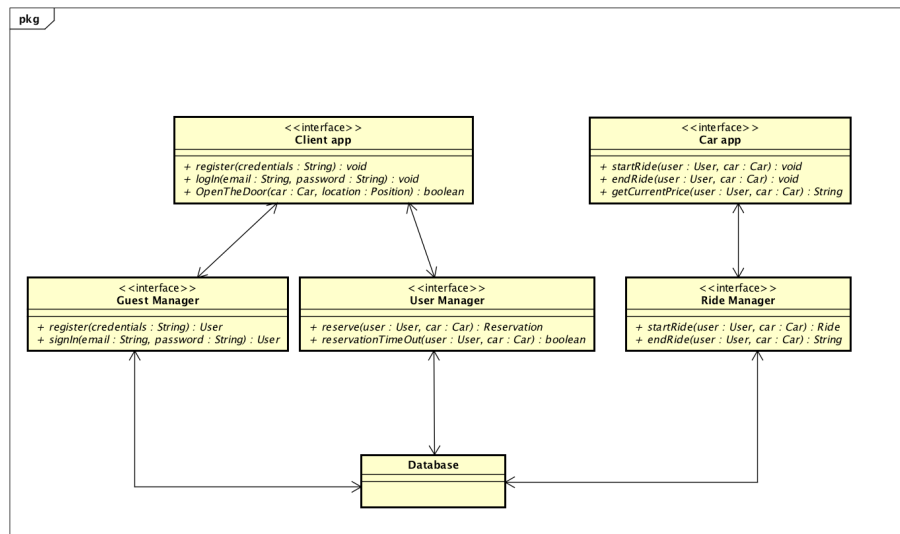
2.10.1 Client app

register(name, surname, email,drivingLisence,paymentInfo)

The register function on the Client app side receives from input all the necessary information for registering. This information is then sent to the server in the format of JSON for former processing.

logIn(email, password)

The logIn function receives from the input the user's email and corresponding password. This information is then sent to the server in the format of JSON for former processing.

car[] getAvailableCars(location)

By providing a location, users can get all the available cars nearby. The location information is sent to the server in the format of JSON then the server sends back all the cars that are available near that location.

void reserve(user, car)

The reserve function allows a user to reserve a car. The reserve request is then sent to the reservation Manager in the server.

void openTheDoor(car,location)

When the user gets to the reserved car, he can inform the server that he is nearby by sending his current location to the server.

2.10.2   Car app

void startRide(user, car)

Car app notifies the server that a ride starts.

void endRide(user, car, state)

Car app notifies the server that a ride is finished.

String getCurrentPrice(user,car)

The working car can get the current price information from the server.

2.10.3   Guset Manager

Boolean register(credentials)

The register function receives from clients the credentials. The Guest Manager in the server checks whether the received credentials already exist in the database or not. In the case of new coming credentials, the

Guest Manager uses Gmail API to send an email to the guest with a newly generated password. Then Guest Manager creates a new tuple using credentials and password and inserts it into the database. In the case of a successful register, a true value will be sent to the client app. In the case of non valid credentials, the server sends back a false notation to the client app.

String signIn(email, password)

The signIn function checks the email and password provided by the user in the server side. If they are valid i.e. there exists a corresponding email and the password matches the email, then the information of the user will be encoded into a JSON format and be sent back to the client app.

2.10.4   User Manager

String reserver(user, car)

User makes request to the server for a car reservation. User Manager processes the request and makes reservation to the corresponding car. The reservation information is then stored into database and the corresponding car information will be sent back to client app in the format of JSON.

Boolean reservationTimeOut(user, car)

When the reserved car is timing out, User Manager notifies the corresponding user and cancels the reservation.

2.10.5   Car Manager

Ride startRide(user,car)

Car Manager on the server side receives from car app the user and car information. It checks from the database whether the ride request is valid or not. In the case of a valid request, server sends back to the client app a successful notation. Also Car Manager stores the data into the database. In the case of a non-valid request, the server sends back to the car app an error notation.

String endRide(user,car,state)

Car Manager processes the request of ending a ride from the car app. It gets the ride information from the database and calculates the final price of the ride. Then Car Manager sends back to car app the price needs to pay. Car Manager redirects payment request to the bank service.

2.11 Selected architectural styles and patterns

2.11.1 Architecture

We choose the three-tier architecture for our system. The whole system is divided into three parts : presentation tier, business tier and data tier.

- presentation tier : It contains Client app and Car app. It is the layer that directly interacts with the actors.

- business tier : Business tier contains all the logic. It manages request and response. Also it manages the data interaction with the database. By using the business tier, clients do not directly interact with the database.

- data tier : We use MySQL database to store all the useful data.
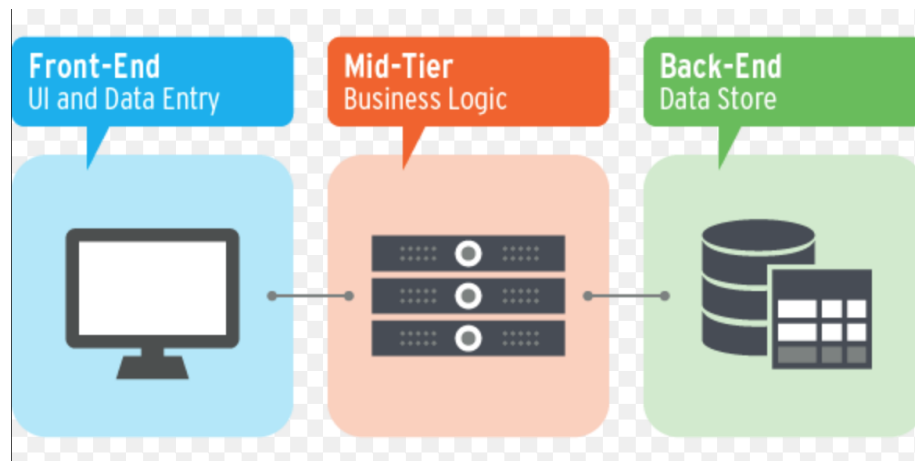
The three tier architecture is used because an effective distributed client-server design is needed. It guarantees increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. In our system, no direct database access from the client app is allowed. All the requests are passing through

business tier first.

We use a typical client-server architecture. In the client-server architecture, data is manipulated in the server side and can only be accessible to designated users. Moreover, services can be easily maintained or updated in the server side without touching Client app.

The Client app is composed of a mobile application and a web browser. Users can choose to access the service by either a mobile web browser or an application. Our system uses request-response methodology. All the services are requested from the user side. The client asks for information or requests for certain services using REST and server responses in JSON format. When the server receives a certain request, it dispatches the request to corresponding responsible manager.

We use RESTFUL API for for our service to be processed. Each service is defined by a unique URI. By using RESTFUL API, we keep our client-server communication clear and simple. Moreover, REST style client-server architecture allows stateless session in the server. All the information transfered between server and client side is in JSON format which allows the information to be processed by different end-systems.

2.11.2   Pattern

MVC Model-View-Controller is used in our system. To combine it with our three-tier architecture, the view part lies on the client app side which deals with interaction with users. Model and Controller are in the server side, which maintains business logic and manipulation of data.

Client-Server In our system, there are 2 clients: user application and car application. The user application can be a pure iOS or Android app or it can be a mobile web browser. In both cases, we use RESTFUL API to communicate with the server. The information sent and received is in JSON format.

Request-response Our system is based on service which uses request-response methodology. All the requests are generated from the user side which can be either a user application or a cat application. Server is responsible for processing the request and sending back the response.

2.12   Other design decisions

The main design architecture and patterns are described above. Just
to mention, our system uses some other service api to meet the require-
ments. For example, we use Google map api to deal with geographical
information. Also all the billing requests are forwarded to other bank
payment systems.

3 ALGORITHM DESIGN

In out system, we have not used any complex algorithms. The request-response methodology is pretty straight forward. In this section, some important interface functions are described in pesudocode.

```
function dispatch(String request) {

String url = parse(request);

switch(url) {

case "register" : forwardTo(Gust Manager);
case "logIn" : forwardTo(Guest Manager);
case "reserve" : forwardTo(User Manager);
case "openTheDoor" : forwardTo(User Manager);
case "startRide" : ForwardTo(Car Manager);
case "endRide" : ForwardTo(Cae Manager);
}
}


function calculateDistance(Position x1, Postion x2}) {

//using Googla Map api to calcualte distance

lag1 = x1.latitude;
lon1 = x1.longitude;
lag2 = x2.latitude;
lon2 = x2.latitude;
distance = calculate(lag1,lon1,lag2,lon2);
}


function getAvailableCars(Position location) {

Map carDistance = new HashMap();
for all car in Cars {
if ()dis = distance(car.position, location)) < nearbyDistance
carDistance.add(car,dis);
}
return car.Distance.sortedByDistance()
}
```

4   USER INTERFACE DESIGN

This section will describe different interfaces for the components. Component interfaces are divided into three different aspects : Guest Manager, User Manager and Car Manager.

5 REQUIREMENT TRACEABILITY

In this section, we are going to show a precise list of all the functional requirements and their corresponding mapping to components in the architecture.(for more detailed view of how these components contact with others and the interfaces of these components, please refer to the high level architecture and the component interface)

- **[G1]**Users can register to the system and get the password by providing credentials and payment information
  - Guest Application Manager
  - Database Manager
  - User

- **[G2]**Registered users can log in by providing the correct credentials
  - User Application Manager
  - Database Manager
  - User

- **[G3]**Registered users can find the locations of available cars within a certain distance from their current location or from a specified address
  - User Application Manager
  - Database Manager
  - Car
  - Area

- **[G4]**Among the available cars in a certain geographical region, users must be able to reserve a single car for up to one hour before they pick it up.
  - User Application Manager
  - Database Manager
  - User
  - Car
  - Reservation
  - Google Map api

- **[G5]**If a car is not pickedup within one hour from the reservation, the system tags the car as available again, and the reservation expires; the user pays a fee of 1 EUR.
  - Car
  - Reservation

- Database Manager
- Car Application Manager
- Bank system

- **[G6]**A user that reaches a reserved car must be able to tell the system shes nearby, so the system unlocks the car and the user may enter.
  - User Application Manager
  - Database Manager
  - Reservation
  - Car
  - Google Map api

- **[G7]**As soon as the engine ignites, the system starts charging the user for a given amount of money per minute; the user is notified of the current charges through a screen on the car.
  - Car Application Manager
  - Database Manager
  - Ride
  - Car

- **[G8]**The user is notified of the current charges through the screen on the car
  - Car
  - Ride
  - Database
  - Car Application Manager

- **[G9]**The system stops charging the user as soon as the car is parked in a safe area and the user exits the car; at this point, the system locks the car automatically.
  - Car Application Manager
  - Database Manager
  - Car
  - Area
  - Google Map api

- **[G10]**The users can know the locations of the safe areas
  - User Application Manager
  - Database Manager
  - Area
  - Google Map api

- **[G11]**If the system detects the user took at least two other passengers onto the car, the system applies a discount of 10% on the last ride.
  - Car Application Manger
  - Database Manager
  - Ride
  - DiscountAndPunish

- **[G12]**If a car is left with no more than 50% of the battery empty, the system applies a discount of 20% on the last ride.
  - Car Application Manger
  - Database Manager
  - Car
  - Ride
  - DiscountAndPunish

- **[G13]**If a car is left at special parking areas where they can be recharged and the user takes care of plugging the car into the power grid, the system applies a discount of 30% on the last ride.
  - Car Application Manger
  - Database Manager
  - Car
  - Area
  - Ride
  - DiscountAndPunish
  - Google Map api

- **[G14]**If a car is left at more than 3 KM from the nearest power grid station or with more than 80% of the battery empty, the system charges 30% more on the last ride to compensate for the cost required to re-charge the car on-site.
  - Car Application Manager
  - Database Manager
  - Car
  - Area
  - Ride
  - DiscountAndPunish
  - Google Map api

- **[G15]**If the user enables the money saving option, he/she can input his/her final destination and the system provides information about the station where to leave the car to get a discount. This station is

determined to ensure a uniform distribution of cars in the city and depends both on the destination of the user and on the availability of power plugs at the selected station.
- User Application Manager
- Database Manager
- Car
- Area
- Reservation
- Google Map api

6   EFFORT SPENT

29/11/2016 ZHOU YINAN 3h DD basic structure and introduction part
30/11/2016 ZHOU YINAN 1h add parts of component interface
2/12/2016 ZHOU YINAN 1h add component interface and diagram
4/12/2016 ZHOU YINAN 2h architecture and design pattern
4/12/2016 ZHAO KAIXIN 4h component view,high level architecture
4/12/2016 ZHAN YUAN 3h add deployment view and part of runtime
5/12/2016 ZHOU YINAN 1h architecture and pattern
5/12/2016 ZHAN YUAN 2.5h add part of runtime view
6/12/2016 ZHOU YINAN 1h document structure modified
6/12/2016 ZHAO KAIXIN 2h requirement traceability
7/12/2016 ZHAO kAIXIN 1h requirement traceability

7   REFERENCES

- RASD v1.0
- Specification Document Assignments AA 2016-2017